MultiMaterial ALE Methods in Unstructured Grids

by

James S. Peery

Daniel E. Carroll

Sandia National Laboratories
Albuquerque, New Mexico 87185-0819

ABSTRACT

Arbitrary Lagrangian Eulerian (ALE) methods have existed for several decades. However, three-dimensional multi-material arbitrary Lagrangian Eulerian (MMALE) methods for unstructured grids are relatively new. MMALE algorithms provide the framework to model sections of a simulation as either Lagrangian, ALE, or Eulerian. In addition, sections of a simulation can switch in time between mesh motions as the distortion of the problem dictates. The MMALE method provides the accuracy of Lagrangian mesh motion and the robustness of Eulerian mesh motion within the same framework. Extending this method to unstructured grids allows for more accurate representation of curved surfaces and complex geometries. This paper examines the algorithms required for the MMALE method and the extensions to these algorithms for unstructured meshes. In addition, second order behavior of the MMALE algorithm as it exists in the high energy density physics code ALEGRA is demonstrated, along with a practical example of its usefulness.

1. Introduction

One driving force for the use of Multi-Material Arbitrary Lagrangian-Eulerian (MMALE) methods at Sandia National Laboratories (SNL) is a critical need to accurately model phenomena and exact geometry associated with the large strain rate requirements of inertially confined fusion (ICF). The ALEGRA code has been developed to apply MMALE in ICF applications. However, the MMALE method in ALEGRA[1] has also been used to model many large strain rate (shock physics driven) events in the areas of shock-activated device performance, weapon safety experiments, and armor and anti-armor applications. In addition, the MMALE algorithms have been formulated to be physics independent and currently support radiation transport, magneto-hydrodynamics, electrostatics and vortex methods for incompressible flow.

The explicit finite element formulation of the governing equations used by ALEGRA results in an inherent time step size limitation, Courant Limit, in order to maintain a stable solution. In the Lagrangian formulation the elements that form the geometry of the problem become distorted and as a result, the time step size must usually be reduced. At some point, the time step size may become too small to economically continue with the simulation. Arbitrary Lagrangian Eulerian (ALE) methods[2] can be used to reduce mesh distortion thus increasing the time step size. The MMALE approach allows for multiple materials in an element and therefore allows for additional flexibility over the traditional ALE method. Mesh distortion can be totally eliminated by using the Eulerian approach. However, Eulerian methods may be too diffusive and are computationally very expensive. The MMALE approach in ALEGRA allows for Lagrangian, Eulerian and smoothing mesh motion within a single framework. Mesh smoothing can approach an Eulerian formulation but has the advantage of being performed less frequently. After moving the mesh (remesh step) a remapping of variables to the new mesh locations is performed. Second order accurate interface reconstruction and advection methods are essential for accurate remapping.

Figure 1 presents a flowchart of the individual modules that make up the MMALE algorithm and a high-level overview of the discussion of this paper. Section 2 describes the data structures that support the use of the MMALE algorithm in an unstructured finite element code such as ALEGRA. Section 3 describes in detail, the remeshing and remapping phases of the MMALE algorithm.

2. Initialization

Within the MMALE method, a number of special data structures and object attributes are required to provide flexibility to the programmer and the user. Finite elements are grouped into "blocks", element faces for boundary conditions are grouped into "side sets", and nodes for boundary conditions are grouped into "node sets". A block must consist of a common element type; however, a block can have a number of different materials. Elements within a block do not have to be topologically connected. ALEGRA adds an additional structure called a *domain* that is a grouping of all blocks.

2.1 Node and Element Pointer Based Topology

A major aspect of any ALE method is the data structures available to easily access node and element neighbor information. In most finite element approaches, the only topological information that is stored is the element connectivity list. Traditional ALE methods have more of a finite difference or finite volume flavor. When ALE methods are added to a finite element code, as we have done, new topology data structures are required. Many of the algorithms that determine new node locations require that a node's closest node neighbors be known. Moreover, methods for remapping element and vertex centered quantities (ex. velocity) require information from neighboring elements. For structured meshes, most ALE formulations use special ordering for node and element neighbors and store these orderings in integer lists.[3] For unstructured meshes, these lists vary in size due to the arbitrary nature of the element-to-element connectivity. ALEGRA uses a more modern approach. Elements and nodes in ALEGRA are C++ objects. Within these objects, pointers to other objects are stored. The "pointer" approach has the same memory and CPU efficiency as the list approach but it is much more flexible and robust. Figure 2, depicts the structure of the "pointer" topology.

2.2 Element-Node Connections

Within ALEGRA, each element stores an array of pointers for the nodes that form its shape. The number of node pointers is determined by the element type (e.g. HEX = 8). The node pointers are stored within each Element object in a predefined order. Each Node object, on the other hand, stores an array of element pointers for the Element objects that are connected to it. The number of element pointers is determined by the connectivity and ranges from 1 to n where n can be arbitrarily large. Due to the arbitrary nature of this array size, a first pass is made through the element topology to determine the number of elements connected to each node. Next, the element pointer array for each Node object is allocated and lastly the memory location for each Element object (i.e., the element pointer) connected to the node is stored in the element pointer array.

2.3 Element-Element Connections

Each Element object in ALEGRA contains an array of Element object pointers for the elements that share a common face. In ALEGRA, this is the definition of an element's element neighbors. For quadrilaterals, an element has at most four element neighbors while in hexahedrals an element has at most six element neighbors. Initially, these arrays are allocated to the maximum number of neighbors and the addresses for the pointers are set to the Element object's own address. Once all the neighbors have been identified, any element pointer that points back to the element itself dictates that the face is a physical boundary of a conformal mesh. This allows for easy tagging of reflective or voided boundaries for the advection logic. Fac-

es for each element are determined from the local canonical node ordering for each element type. Element neighbors for each element are tagged as {left, front, right, back, bottom, top} faces. From this ordering, it is obvious which face is opposite to a given face and thus one can find a neighbor's neighbor which is required for some advection algorithms.

2.4 Node, Element, Block, and Domain Attributes

The ALE algorithms in ALEGRA are optimized for CPU and memory efficiency. To accomplish this, each Node and Element object contain attributes that are used throughout the remesh/remap algorithm. In addition, a number of attributes are specified by the user for blocks and the domain which provide a range of controls. Through the initialization phase, the user specified block and domain attributes help determine many of the element and node attributes. For example, each block of elements can be specified as Lagrangian, SMALE, MMALE, or Eulerian. Each ALE block of elements has user defined "triggers" for tagging nodes to be moved in order to reduce the distortion in the mesh, sometimes referred to as mesh smoothing. Eulerian blocks can specify the directions to be Eulerian leaving unspecified directions to behave in a Lagrangian fashion.

The Node Type attribute for each node is set through a combination of the Block and Domain attributes. The algorithm uses the order of block, face (side set), and edge (node set) in determining a node's type along with the inequality provided in Equation (1). In other words, for a single node, a setting derived from an element block will be overwritten by a derived setting for a side set which in turn can be overwritten by the edge setting that applies to the node. In addition, some nodes will be located on interfaces between blocks, side sets and node sets that have different Mesh Type settings. To properly handle these cases, the following precedence is used:

$$\text{Lagrangian} > \text{SMALE} > \text{MMALE} > \text{Eulerian} \tag{1}$$

3. MMALE Methods and Algorithms

The MMALE algorithms in ALEGRA involve identifying nodes that need to be moved based on distortion criteria, relocating nodes to relieve the detected distortion, and remapping the element and node variables while conserving global quantities. These steps and their substeps are described in the paragraphs that follow.

3.1 Remesh

The remeshing phase of the ALE method requires determining new locations for the nodes that would partially alleviate their associated elements' distortion. Figure 3 demonstrates how Lagrangian, ALE and Eulerian nodes behave through the remesh step. Eulerian nodes are moved back to their position after the Lagrangian step and Lagrangian nodes are not considered for movement. However, remeshing of ALE nodes involves three steps. After the Lagrangian step, a list of nodes that meet a user specified distortion criteria for being moved is created. Next, new positions are calculated for this list of nodes. Lastly, the actual movement of the selected nodes is limited to some fraction of their calculated movement, for reasons specified below. The details of these three steps are presented in the following subsections.

3.1.1 Criteria for Moving an ALE Node

ALEGRA's criteria for moving a node is that given by Barton in the HEMP[4] code as extended by Benson. The criteria listed by Barton can be summarized with two tests: an angle test and a volume test. For every node, the angles that are formed by the vectors given by element edges that originate at the node are calculated. The ideal situation would be for all of the angles to be 90 degrees. The calculation of the angles begins by determining the length of the sides and the area of the triangles formed in each element by the node and its two closest nodes. An example of the triangle formed by two vectors with the vertex at node 1 in shown in Figure 4.

The angle θ is given by[5]

$$\sin\theta = 2\frac{A}{bc} \text{ or } \theta = \text{acos}\left[\frac{\hat{b} \bullet \hat{c}}{bc}\right] \tag{2}$$

and the area of triangle **abc** is given by[5]

$$A = \sqrt{s(s-a)(s-b)(s-c)} \tag{3}$$

where

$$s = \frac{1}{2}(a+b+c) \tag{4}$$

These calculations are performed for each of the elements that surround the node. In three-dimensions, each of the three faces that connect to a node for a given element are checked. To determine if a node should move the following minimum angle (maximum of the cosine of the angle) is used as a measure of distortion and is given as:[4]

$$R^{\theta} = max(\cos\theta_1, \cos\theta_2, \cos\theta_3, \ldots) \tag{5}$$

where the list of angles represents the angles formed by edges around a node. A volume measure of distortion is found by comparing the minimum to maximum volumes of all elements that are connected to a node. This is given as

$$R^{A} = \frac{min(V_1, V_2, V_3, \ldots)}{max(V_1, V_2, V_3, \ldots)} \tag{6}$$

The user selects the cutoffs for Equations (5) and (6). A user specified setting of zero or cosine(90) for Equation (5) will cause every node to be selected for remeshing within a block. Likewise, a user specified setting of 1.0 for Equation (6) will cause every node to be selected for remeshing. Note that due to the behavior of the cosine function, Equation (2) is valid for both acute and obtuse angles if the absolute value is used.

3.1.2 Logic for tagging nodes that are Lagrangian, Eulerian, or ALE

When a node is determined to be remeshed, elements and nodes that are topologically connected to the node are affected. These elements and nodes will participate in the advection schemes and processing only these objects is an important optimization. If all nodes are tagged to be moved, all elements and nodes are tagged as AFFECTED. This is the case for either pure Eulerian calculations or in a user defined re-smoothing of the entire mesh. Otherwise, if a node is tagged to be moved and it is an ALE node, all of the elements attached are tagged as being ALE elements. Lastly, all nodes attached to elements who are tagged as AFFECTED are also tagged as being AFFECTED.

With all elements and nodes tagged with the proper attributes, the new location for the nodes must be determined. The updating of coordinates is shown in Figure 3 The vector, **New** is used to hold the new node locations and has been initialized to the value of **Current** from the previous time step. For Eulerian nodes, **New** is set to the location from the last remap. Most of the time this is the original location. Before nodes are moved, special cases must be dealt with. ALE nodes set **New** to **Current**. For Eulerian nodes in only one direction, only the directional component of **New** is set to **Current** from the last remap. At this point, ALE nodes can be smoothed and the **New** vector will hold the new coordinates.

3.1.3 Mesh Smoothing

In order to perform the remeshing phase of the ALE technique, an algorithm must be used that determines where a node is to be moved. There are many techniques for deciding where to move a node. ALEGRA contains methods based on equipotential solutions, a technique based on averaging element centers, and a method by Budge[6] that attempts to preserve mesh orthogonality. Of these methods, only the equipotential solver will be discussed in detail.

3.1.3.1 Applying Equipotential Solutions to the Mesh

One of the most successful mesh smoothing methods and the primary method used in ALEGRA is the method based on Winslow's work with equipotential smoothing.[7] Winslow's method is based on inverting Laplace's equation. If one were to generate a grid based on this method, the grid lines would be the characteristic lines of Laplace's equation. In addition, this method very aggressively moves nodes in the attempt to equalize element volumes.

As previously mentioned, the remapping algorithm is based on equipotentials generated by solving the inverted Laplace's equation. This is given as[7]

$$\alpha x_{\phi\phi} - 2\beta x_{\phi\varphi} + \gamma x_{\varphi\varphi} = 0 \tag{7}$$

$$\alpha y_{\phi\phi} - 2\beta y_{\phi\varphi} + \gamma y_{\varphi\varphi} = 0 \tag{8}$$

where

$$\alpha = x_\varphi^2 + y_\varphi^2 \tag{9}$$

$$\beta = x_\phi x_\varphi + y_\phi y_\varphi \tag{10}$$

$$\gamma = x_\phi^2 + y_\phi^2 \tag{11}$$

These equations have been used for most two-dimensional logically rectangular mesh ALE codes. Using second order central differencing techniques, Equations (7) through (11) can be simplified to give

$$x' = \frac{1}{2(\alpha+\gamma)}\left(\alpha(x_4 + x_8) + \gamma(x_2 + x_6) - \frac{1}{2}\beta(x_3 - x_5 + x_7 - x_9)\right) \tag{12}$$

$$y' = \frac{1}{2(\alpha+\gamma)}\left(\alpha(y_4 + y_8) + \gamma(y_2 + y_6) - \frac{1}{2}\beta(y_3 - y_5 + y_7 - y_9)\right) \tag{13}$$

where

$$x_\phi = \frac{1}{2}(x_4 - x_8) \tag{14}$$

$$x_\varphi = \frac{1}{2}(x_2 - x_6) \tag{15}$$

$$x_{\phi\varphi} = \frac{1}{4}(x_3 - x_5 + x_7 - x_9) \tag{16}$$

and equations for the y coordinate are similiarly developed.

Equations (12) through (13) only work for nodes surrounded by four elements and can be easily extended to work in three dimensions for nodes only surrounded by eight elements. This can be a serious limitation for unstructured mesh codes. To overcome this limitation, Robert Tipton of Lawrence Livermore Laboratory, reformulated Winslow's approach using a variational approach.[8] To date, Tipton's derivations remain un-published; however, a summary of his approach is given below. Tipton's algorithm forms the foundation of the smoothing algorithms in ALEGRA.

Tipton begins with the integral

$$I = \frac{1}{2}\int dx^3 W(x)\left|\vec{\nabla f}\right|^2 \tag{17}$$

Using the variational principle, Equation (17) can be shown to result in the following variational equation

$$\frac{\delta I}{\delta f} = -\nabla \bullet W \nabla f = 0 \tag{18}$$

which Tipton demonstrates is the correct form of Laplace's equation in variational form with an arbitrary weight function *W(x)*. Using finite element formalisms, *f(x,y,z)* can be represented in an expansion of shape functions:

$$f(x, y, z) = \sum_{\alpha} f_{\alpha} N_{\alpha}(x, y, z) \tag{19}$$

Substituting Equation (19) into Equation (17) yields

$$I = \frac{1}{2}\sum_{\alpha\beta} f_{\alpha} f_{\beta} \int dx^3 W \nabla N_{\alpha} \bullet \nabla N_{\beta} = \frac{1}{2}\sum_{\alpha\beta} M_{\alpha\beta} f_{\alpha} f_{\beta} \tag{20}$$

where $M_{\alpha\beta}$ represents the classical stiffness matrix with an arbitrary weight function. Using Equation (18), the derivative of I with respect to each f_{α} must vanish which results from Equation (20) as

$$\frac{\partial I}{\partial f_{\alpha}} = \sum_{\beta} M_{\alpha\beta} f_{\beta} = 0 \tag{21}$$

As with the Winslow method, Equation (21) is solved iteratively. Expanding Equation (21) and solving for f_{α}, one gets

$$f_{\alpha} = -\frac{1}{M_{\alpha\alpha}} \sum_{\beta \neq \alpha} M_{\alpha\beta} f_{\beta} \tag{22}$$

Continuing with the finite element approach, $M_{\alpha\beta}$ is discretized for elements and becomes

$$M_{\alpha\beta} = \sum_{el} M_{\alpha\beta}^{el} = \sum_{el} \int_{el} dx^3 W \nabla N_{\alpha}^{el} \bullet \nabla N_{\beta}^{el} \tag{23}$$

At this point, no distinction has been made with respect to dimensionality or element type. The shape functions for the element will determine these attributes. The use of shape functions requires a transformation from Euclidean coordinates $\{x_i\} = \{x, y, z\}$ to isoparametric coordinates $\{\xi_i\}$. This transformation changes Equation (23) to

$$M_{\alpha\beta}^{el} = \int_{el} d\xi^3 W J \sum_{ij} \tilde{g}_{ij} \frac{\partial N_{\alpha}}{\partial \xi_i} \frac{\partial N_{\beta}}{\partial \xi_j} \tag{24}$$

where $\tilde{g}_{ij}$ represents the twice-contravariant metric tensor for the transformation from $\{x_i\}$ to $\{\xi_i\}$ and J is the Jacobian. For convenience, define

$$a_{ij} = J^2 \tilde{g}_{ij} \text{ and } W_{el} = w_{el} ||J|| \tag{25}$$

Given a set of shape functions, Equation (24) can be determined for each element and assembled into Equation (22) for new node positions. Since node positions are needed for Equation (24), a Jacobi iteration procedure is used.

In solving Equation (24) for quadrilaterals, the classical checker-board mode must be resolved. This unphysical mode is the result of having four degrees of freedom in the physical space while the solution space for Laplace's equation has only three. Tipton suggests setting the fourth eigenvalue of the matrix represented by Equation (24) to

$$\lambda_4 = a_{11} + a_{22} \tag{26}$$

With this selection, the stiffness matrix becomes

$$M_f = \frac{1}{2} w_{el} \begin{bmatrix} a_{11} - a_{12} + a_{22} & -a_{22} & a_{12} & -a_{11} \\ -a_{22} & a_{11} + a_{12} + a_{22} & -a_{11} & -a_{12} \\ a_{12} & -a_{11} & a_{11} - a_{12} + a_{22} & -a_{22} \\ -a_{11} & -a_{12} & -a_{22} & a_{11} + a_{12} + a_{22} \end{bmatrix} \tag{27}$$

which, as Tipton points out, is the well known and very successful Pert[9] diffusion operator. Equation (27) is used very effectively in ALEGRA for two-dimensional smoothing.

For hexahedrals, the solution of Equation (24) is more complicated, since there are four physical modes and four unphysical modes. Tipton suggests setting the last four eigenvalues to

$$\begin{aligned} \lambda_5 &= a_{11} + a_{22} + a_{13} + a_{23} \\ \lambda_6 &= a_{22} + a_{33} + a_{12} + a_{13} \\ \lambda_7 &= a_{11} + a_{33} + a_{12} + a_{23} \\ \lambda_8 &= a_{11} + a_{22} + a_{33} \end{aligned} \tag{28}$$

With these selections, the stiffness matrix for the hexahedral element becomes

$$M = \frac{w_{el}}{4} \begin{bmatrix} m_{11} & -a_{11} & -a_{12} & -a_{22} & -a_{33} & -a_{13} & 0 & -a_{23} \\ & m_{22} & -a_{22} & a_{12} & a_{13} & -a_{33} & -a_{23} & 0 \\ & & m_{33} & -a_{11} & 0 & a_{23} & -a_{33} & a_{13} \\ & & & m_{44} & a_{23} & 0 & -a_{13} & -a_{33} \\ & & & & m_{55} & -a_{11} & -a_{12} & -a_{22} \\ & & & & & m_{66} & -a_{22} & a_{12} \\ & & & & & & m_{77} & -a_{11} \\ & & & & & & & m_{88} \end{bmatrix} \tag{29}$$

where taking advantage of the fact that the unit vector is in the null space of M,

$$m_{ii} = -\sum_{j=2}^{8} m_{ij} \tag{30}$$

For diagonal coupling of nodes 1 and 7, Tipton suggests setting the last four eigenvalues to

$$\begin{aligned} \lambda_5 &= a_{11} + a_{22} \\ \lambda_6 &= a_{22} + a_{33} \\ \lambda_7 &= a_{11} + a_{33} \\ \lambda_8 &= a_{11} + a_{22} + a_{33} \end{aligned} \tag{31}$$

Both Equations (28) and (31) are available in ALEGRA.

Finding the coordinates, $\{x_i\}$, for all the nodes that satisfy Laplace's equation is an iterative procedure. Thus, Jacobi iteration is used to solve Equation (24). In ALEGRA, the user is allowed to control the number of iterations performed on Equation (24). Generally, ALE algorithms are only interested in making small changes to the mesh and therefore the exact solution to Equation (24) is not desired since it may violate a Courant limit by moving the nodes to far.

3.1.4 Mesh Smoothing Examples

There are a multitude of smoothing combinations for both interior and exterior nodes. In addition, the behavior of nodes that lie among different mesh types (Lagrangian, SMALE, MMALE, Eulerian) can greatly influence the results of smoothing operations. Examples are provided here for the equipotential Tipton smoother, both with and without boundary smoothing options activated. Figure 5 displays the initial 10x10 graded rectangular mesh, while Figure 6 shows the result of 100 iterations of the Tipton smoother. The algorithm is attempting to equalize volumes and in this case, the Tipton smoother is identical to the Winslow solution. Boundary constraints in the form of non-moving boundary nodes prevent the method from equalizing volumes.

3.1.4.1 Boundary Node Smoothing Options

ALEGRA is designed to allow the user to specify node smoothing behavior on ANY node set. In contrast, unless a user specifies a boundary as being ALE, it cannot be smoothed. For the smoothing to work well, the nodes in a node set must fall on a straight line for edges, or on a xy, xz, or yz plane for faces. ALEGRA cannot currently smooth curved surfaces.

Unless specified otherwise, boundary nodes are defined to be Lagrangian in LAGRANGIAN, SMALE, and MMALE meshes, and Eulerian in EULERIAN meshes. The user can specify the behavior of ANY node set as LAGRANGIAN, ALE, or EULERIAN. Most of the time the user will want to make these specifications on exterior boundaries, however it is possible to run an entire simulation as Lagrangian and have one node

anywhere be tagged as ALE.

For edges, the smoothing algorithm attempts to equally distribute the nodes between the end-points. Faces are smoothed with the two-dimensional variations of the smoothers described above. ALEGRA uses the order of interior, face and edge in applying the smoothing techniques. Due to the aggressive behavior of smoothing combinations, the user can control the amount of nodal movement that results in the remesh step.

3.1.4.2 Boundary Node Smoothing Examples

ALEGRA is designed to allow the user to specify node smoothing behavior on ANY node set. This can greatly relieve the distortion in the mesh. But, the current algorithms in ALEGRA are limited to straight lines and planar faces. In Figure 7, only the line Y=0 is specified as ALE. Within this designation, only the nodes within the end-points are allowed to move. The algorithm for moving these nodes basically moves the nodes to the middle of the two neighbor nodes, forcing the nodes to become equally spaced. In comparison with Figure 6, one can see that the mesh in Figure 7 is becoming more optimal with the relaxation of the y=0 boundary. In Figure 8, all of the boundary nodes are specified as ALE nodes and the results are more dramatic. Currently, the code does not allow for curved lines or corners to be smoothed. These features will be added in the future and it is possible that these features can be added so that conservation is maintained for most simulations. Aggressive boundary smoothing will require knowledge or construction of the deformed geometry.

3.1.4.3 Smoothing examples in SMALE and MMALE blocks.

This section demonstrates how nodes connected among a mixture of mesh types behave during remeshing. These nodes are called interface nodes. The general rule is that an interface node is LAGRANGIAN if it is connected to a Lagrangian or SMALE mesh, ALE if connected to either all MMALE meshes or a mixture of MMALE and EULERIAN meshes, and Eulerian if connected to only EULERIAN meshes. The next sections will provide examples to better clarify the behavior of interface nodes under mixed mesh type conditions. The examples below begin with the initial conditions shown in Figure 5; however there are four mesh blocks that make up the domain. Meshes (blocks) one through four have a common point at x = 0.5 and y = 0.5 and the mesh blocks 1 through 4 are located in quadrants 3, 4, 1, and 2, respectively. In addition, all boundaries are flagged as ALE.

In the first example, a four mesh block problem, where each mesh block is skewed, is modeled as four SMALE meshes. Based on the above discussion, the interface nodes will behave Lagrangian and thus cannot be smoothed. This behavior is depicted in Figure 9 where one can see that each individual mesh is smoothed where as the interface nodes are held in place throughout the smoothing operation. In the next example, the behavior of an interface node connected to a SMALE mesh and a MMALE mesh is investigated. In Figure 10, mesh 1 is SMALE while meshes 2 through 4 are MMALE. As expected, the interface nodes between the MMALE meshes are smoothed while the interface nodes between

the SMALE mesh and the MMALE meshes are Lagrangian and thus are not smoothed. The last example shows the behavior of nodes between MMALE and EULERIAN meshes. In Figure 11, mesh 1 is Eulerian while meshes 2,3, and 4 are MMALE. As one can see, the interface nodes are smoothed.

3.1.4.4 Three Dimensional Smoothing Example

Effective smoothing in three-dimensional problems requires more consideration of the effects of boundaries. Visualizing the effects of three-dimensional interior smoothing is difficult and without boundary smoothing can be very constrained. Many three-dimensional problems have planar surfaces that align with the xy, xz or yz planes. Applying interior and boundary surface smoothing can be very effective for problems that have these features. One such example is shown in Figure 12. In this problem, a plasma annulus is compressed toward the axis of symmetry. A voided region exists in the interior of the plasma annulus. Without mesh smoothing the voided region cannot “get out of the way” of the impinging plasma and the problem would in effect stop due to the collapse of the outer ring of voided elements. To facilitate the simulation of the problem, Tipton smoothing is performed on the interior of the volume and on both xy planar surfaces. Without weights, Tipton smoothing would push the nodes outward which would be worse than running without smoothing. However, an inverse radius weight is used in conjunction with the Tipton smoother which in effect pulls the voided nodes toward the axis of symmetry. This approach has proven very effective in running magnetically driven axial plasma pinches (z-pinches).

3.1.5 Limiting Node Movement

As mentioned in the discussion above, solution to the mesh smoothing equations is obtained by an iterative approach. During the iterations, nodes can be moved in a manner that causes an element to “over empty” and thus result in a negative volume. Moreover, overly aggressive node movement can cause unnecessary diffusion, due to interpolation errors inherent in the advection method. To prevent these undesirable effects, the user can control the percent of node movement from its original location to the location determined from the smoothing.

3.2 Remap

The remap step in ALEGRA uses second-order advection methods for both node and element variables, high resolution interface reconstruction and one-dimensional passes. Currently, ALEGRA performs a remap on unstructured quadrilateral and hexahedral meshes. The remap step involves several substeps that include:

1) Determine volume fluxes

2) Advect element centered non-material variables

3) Advect node centered volume dependent variables

4) Determine material fluxes via interface reconstruction

5) Advect element centered material variables

6) Advect node centered mass dependent variables

Steps 2 through 6 are performed in sequence as alternating direction one-dimensional passes. This approach fosters proper corner coupling, the movement of material diagonally in the mesh. In the sections that follow, the algorithms and methods used in steps 1 through 6 will be described in detail. In addition, the effects of an unstructured mesh on these methods will be discussed.

3.2.1 Determining Volume Fluxes

The first step in any advection scheme is to determine the volume fluxes created by nodal movement. When a node is moved, volume fluxes are generated through the sides of the elements to which the node is attached. The volume flux through an element side is given by the volume of the quadrilateral in two dimensions, or the hexahedron in three dimensions, that is created from the set of old node positions and new node positions on every face of the element. The volume flux is calculated very much like the volume of an element since the CURRENT and NEW coordinates of an element's face form a sub-element. For hexahedrons, the volume created by node movement can have bi-linear faces and thus the volume must be calculated in the same way as a finite element hexahedron volume. The volume of an element, using finite element terminology, is given as

$$V^e = \int_{\Omega_e} dV = \int_{-0.5}^{0.5}\int_{-0.5}^{0.5}\int_{-0.5}^{0.5} J\,d\xi\,d\eta\,d\zeta \tag{32}$$

where

$$J = \det\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \varepsilon_{ijk}\frac{\partial x}{\partial \xi_i}\frac{\partial y}{\partial \xi_j}\frac{\partial z}{\partial \xi_k} \tag{33}$$

is the Jacobian of the transformation from the isoparametric coordinates ξ_i to the current physical coordinates x_i. Now

$$x_i = x_{iI}^e N_I \tag{34}$$

where x_{iI}^e are the coordinates of the element face vertices before and after node movement. The N_I are the element shape functions used to represent variable variation within the element. Thus

$$\frac{\partial x_i}{\partial \xi_j} = x_{iI}^e \frac{\partial N_I}{\partial \xi_j} \tag{35}$$

and

$$J = \left(\varepsilon_{ijk} \frac{\partial N_I}{\partial \xi_i} \frac{\partial N_J}{\partial \xi_j} \frac{\partial N_K}{\partial \xi_k} \right) x_I^e y_J^e z_K^e = c_{IJK} x_I^e y_J^e z_K^e \tag{36}$$

Then

$$V^e = \int_{-0.5}^{0.5} \int_{-0.5}^{0.5} \int_{-0.5}^{0.5} c_{IJK} x_I^e y_J^e z_K^e d\xi d\eta d\zeta = C_{IJK} x_I^e y_J^e z_K^e \tag{37}$$

In principle, the matrix C_{IJK} has 512 components. In fact, the number of independent components is much smaller, since the matrix is fully antisymmetric. Antisymmetry reduces the independent component count to 168. Furthermore, since one can choose any node in the element as the first node to number, and any of three others as the second, the count is further reduced to 7. These components are

$$\begin{aligned} C_{123} &= -\frac{1}{12} & C_{127} &= 0 \\ C_{125} &= \frac{1}{12} & C_{128} &= 0 \\ C_{126} &= \frac{1}{12} & C_{135} &= 0 \\ & & C_{136} &= 0 \end{aligned} \tag{38}$$

This permits efficient calculation of the element volume fluxes for three-dimensional simulations. In two-dimensional calculations, the volume flux is easily calculated from the quadralateral area.

Equation (37) defines the volume flux such that a positive volume flux will occur for an element side if the node is moved further into the element's region. In other words, the volume flux is assumed to be positive if the volume is leaving the element. As an optimization, negative volume fluxes are set to zero unless they occur on a physical boundary. This optimization eliminates half of the remap operations and prevents double counting errors. As a further optimization, volume fluxes are only calculated for elements marked as AFFECTED by node movement.

With the definition of volume flux given by Equation (32), all fluxes in the neighbors are added to an element's volume while the fluxes from the element are subtracted. With this procedure, the new volume of an element is given by

$$V_{new} = V_{old} + \sum_{i=1}^{en} (\Delta V_{ni} - \Delta V_i) \tag{39}$$

where *i* refers to an element's side, *ni* refers the corresponding neighbor element's side and *en* is the number of element neighbors. The new volume could also be calculated from the new node coordinates. In ALEGRA, Equation (39) is applied in one-dimensional pass-

es, ias explained below. As volumes are subtracted from an element, they are added to its neighbor.

Boundaries on a mesh can be specified as either voided or reflective (default). In both cases, negative volume fluxes are saved. For voided boundaries, void is fluxed into the element and for reflective boundaries, the volume flux is partitioned in the ratio of materials contained in the element.

3.2.2 General Advection Principles

Isotropic advection assumes that the material is advected through all element faces simultaneously. Without a very careful geometric implementation using both element face and diagonal neighbors, this approach will fail to advect quantities correctly when the material flow is not aligned with the element faces and is considered a low order method. To overcome this limitation, ALEGRA uses the concept of one-dimensional passes with a permutation that switches the order of the one-dimensional passes each advection cycle. This procedure provides for corner coupling; however, as opposed to a logically rectangular mesh, a finite element mesh does not have a predefined set of directions. In order to overcome the lack of logical (i,j,k) direction, ALEGRA works in logical space by predefining element face pairs. This approach has been shown to be second-order where the grid is logically rectangular and first-order where the grid is unstructured (e.g. three elements sharing a node). Such an approach is much more accurate than using an isotropic method everywhere and, moreover, finite element meshes are mostly logically rectangular with isolated areas of unstructured topology.

In effect, one-dimensional passes view the elements aligned in strips where the “left” and “right” advection volumes are processed along with remapping the variables before the “top” and “bottom”, “front” and “back” advection volumes are processed. This process is depicted in Figure 13. This corner coupling is not exact but is close enough that most ALE formulations avoid the alternative nine element stencil. The permutation in direction for each cycle ensures that a given direction is not preferred. Examples of corner coupling will be discussed in section 3.2.3.

Advection algorithms can be expressed in volume or mass coordinates depending on the nature of the variable. For example, stress is defined in terms of per unit volume, while specific internal energy is given in terms of per unit mass. For a volume based variable, a newly advected element centered variable is given by

$$f_{new} = \frac{f_{old}V_{old} + \sum_{i=1}^{2} \tilde{f}_i(\Delta V_{ni} - \Delta V_i)}{V_{old} + \sum_{i=1}^{2} (\Delta V_{ni} - \Delta V_i)} \tag{40}$$

where the summation is for both faces of the element in a one dimensional pass and the $\tilde{f}_i$ are determined by the type and order of the advection algorithm. Note that in Equation (40), for a given element face, either ΔV_{ni} or ΔV_i will be zero since negative internal advection volumes have been zeroed. For mass based variables, Equation (40) becomes

$$f_{new} = \frac{f_{old}M_{old} + \sum_{i=1}^{2} \tilde{f}_i(\Delta M_{ni} - \Delta M_i)}{M_{old} + \sum_{i=1}^{2} (\Delta M_{ni} - \Delta M_i)} \tag{41}$$

Equations (40) and (41) work for material variables with the appropriate changes to material volumes or masses and fluxes. From this point on, the equations will be derived for volume-weighted quantities, keeping in mind that the advected quantities may be mass-weighted.

More research effort has been expended in finding approximations to $\tilde{f}_i$ than any other area of ALE algorithms. For element centered advection, ALEGRA may use donor cell, Van Leer[10], and SuperB[11] schemes. Donor cell is the simplest advection method and is a first-order since it does not involve evaluating derivatives within an element. The flux is assumed to carry the value of any element-centered variable from which the material came. This is analogous to first-order upwinding in finite difference methods. The most common higher accuracy element-centered advection method used in ALEGRA is Van Leer. Van Leer advection can be shown to be both second order and monotonicity preserving. The method requires information from elements ahead and behind the donor element. As shown in Figure 14, four slopes can be measured from the element variables plotted in volume or mass coordinates.

In the Van Leer stencil, the slope, s_{vl} is given as

$$s_{vl} = min(slope_1, slope_2, \frac{slope_3 + slope_4}{2}) \tag{42}$$

where

$$\begin{aligned} slope_1 &= 2\left\{\frac{f_A - f_D}{V_D}\right\} \\ slope_2 &= 2\left\{\frac{f_D - f_B}{V_D}\right\} \\ slope_3 &= 2\left\{\frac{f_A - f_D}{(V_D + V_A)}\right\} \\ slope_4 &= 2\left\{\frac{f_D - f_B}{(V_D + V_B)}\right\} \end{aligned} \tag{43}$$

If the slopes have different signs, then the slope is set to zero. In effect, sign consistency preserves monotonicity by determining if a maximum or minimum exists in the donor cell. From Equation (42), one can see that the Van Leer algorithm uses the smallest slope. With the slope determined, the fluxed value is given as

$$\tilde{f}_i = \begin{cases} f_i^d + \frac{s_{vl}}{2} V - |\Delta V| & \text{if} \quad \text{sgn}(slope_1) = \text{sgn}(slope_2) \\ f_i^d & \text{if} \quad \text{sgn}(slope_1) \neq \text{sgn}(slope_2) \end{cases} \tag{44}$$

3.2.3 Material Volume Fluxes

In a multi-material cell code, material volume fluxes must be determined from the cell volume fluxes. For example, if a cell contains both air and water, an algorithm must be created that can determine if the cell volume flux contains air, water or a combination of both. This requirement is driven by the lack of individual material velocity fields. Algorithms to determine material volume fluxes fall in the category of material interface reconstruction. ALEGRA is unique in that it contains not only the Simplified Line Interface Construction[12] (SLIC) algorithm but also a version of the high resolution algorithm, Sandia Modified Young's Reconstruction Algorithm[13] (SMYRA) for unstructured meshes. The SLIC algorithm is only used in ALEGRA for comparisons with SMYRA.

The SMYRA algorithm is an adaptation of the original Young's Material Interface Reconstruction Algorithm[13]. The SMYRA method offers the advantage of "second-order" performance over other methods of tracking material interfaces in computational elements with mixed materials. By resolving the possible patterns of materials in the elements surrounding a donor element and dealing with each case on an ad hoc basis, the algorithm allows modeling of interface planes that are not aligned with the directions associated with the normals to the element faces. Also the method allows a treatment of the order in which materials should leave or enter a fluxing volume and thus allows a better representation of the spatial distribution of material in the neighborhood of the element. With the use of SMYRA, objects may be moved across the computational domain with little dispersion, provided they are resolved sufficiently in the first place, i.e three to four elements across "thin" structures.

Application of SMYRA for unstructured meshes requires a significantly different stencil than for structured meshes. In both cases, the algorithm is applied in a one-dimensional sense where fluxes on opposite faces are considered. This is a result of the one-dimensional passes used in the remap step. For unstructured meshes, the SMYRA algorithm uses volume fractions from the upstream and downstream cells along with averaged volume fractions at the vertices of the element. This is depicted in Figure 15 for a two-dimensional grid. The vertex centered volume fractions are determined by averaging the material volume fractions from the surrounding element. From this stencil, SMYRA approximates not only the order of the materials in a cell, but also the interfaces and their normals between the materials. This information is then used by SMYRA to determine the material volumes constrained in the volume flux. The SMYRA algorithm has been shown in two and three dimensions to be superior to the SLIC algorithm and thus is termed a high resolution interface reconstruction algorithm

A classical test of interface reconstruction algorithms is the "balls and jacks" problem proposed by McGlaun.[14] This problem consists of pure advection of two jacks and two

spheres along a diagonal of a grid, as shown for two and three dimensional grids in Figures 16 through 17, respectively. As seen in Figure 16, one of the spheres is made up of concentric balls of different materials. Both two dimensional and three-dimensional results of this problem for structured and unstructured meshes are described below.

3.2.3.1 Two-Dimensional “Balls and Jacks” simulations

In two dimensions, the balls are defined by circles and the jacks by crosses of material. Figures 18 through 19 show the final configurations for the balls and jacks problem for both the SLIC and SMYRA algorithms on a structured mesh consisting on 200 by 200 cells. As expected and demonstrated in a number of hydrodynamic codes, the SLIC algorithm turns the balls into squares and badly distorts the crosses. SMYRA on the other hand, preserves the ball configurations and only slightly distorts the crosses. Figure 20 shows a very unstructured mesh consisting of almost 40,000 cells or approximately the same resolution as the structured mesh. This grid was also used for the two-dimensional balls and jacks problem. Figures 21 through 22 demonstrates the effect of these two algorithms used with the unstructured mesh. As one can see, the SMYRA results are superior to the SLIC results. Also of interest is that both methods give better solutions on the unstructured mesh than the structured mesh. This is an artifact of the fact that more cells are aligned in the direction of flow with the unstructured mesh.

3.2.3.2 Three-Dimensional Balls and Jacks

Figures 23 through 27 demonstrate the SLIC and SMYRA algorithms for structured and unstructured meshes. The structured mesh consists of 100x100x100 cells and as expected the SLIC algorithm elongates the jacks and turns the balls into boxes. The SMYRA algorithm preserves the balls and only slightly distorts the crosses. The unstructured mesh was created by generating 282,000 tetrahedral elements and dividing each tetrahedral into four hexahedral or a total of 1.1 million hexahedral - in effect the same resolution as the three-dimensional structured mesh. As seen in two-dimensions, the SMYRA algorithm is superior to SLIC on unstructured meshes. However, the amount of distortion caused by the SLIC algorithm is markedly less than the case with the structured mesh. This effect is due to the distribution of the orientation of the faces of the elements in this mesh and the smoothing effect that results on advected quantities. Figure 27 provides some appreciation of the degree of non-orthogonality in the unstructured 3D mesh examples. In this “cropped” view of the computational space, the balls and one of the jacks are seen extruding from the elements and the irregularity of the mesh is visible.

3.2.4 Material Variable Advection

Being a multi-material cell code, ALEGRA stores density, specific internal energy, rotation, stretch, stress and state variables for each material that occupies a cell. All of these variables must be advected in the remap step for each material. One can debate what metric should be used in advecting non-conserved quantities (ex. rotation), however, most agree that these quantities must be advected in order to preserve an accurate solution. ALEGRA allows all material variables to be advected with either mass or volume with the element-centered advection methods described above being applied. In this procedure, variables remapped with volume are processed first, with density being the last variable remapped. Applying this order allows the

material volume fluxes to be replaced with material mass fluxes. Next, variables remapped with mass are processed (ex. specific internal energy).

There are several subtle issues associated with material variable advection. The remap process of density and specific internal energy is guaranteed to conserve mass and internal energy. However, the remap of material rotation and stretch can cause irregularities. Rotation is an orthonormal tensor. Through the remap step, the tensor can deviate from this property. Care must be taken to return this tensor's orthonormal property after the advection step. Stretch, on the other hand is symmetric positive definite (SPD) tensor. Numerical results have demonstrated that due to both round-off and application of a mixture of first and second order advection stencils to the components of the tensor, a non-SPD tensor can result. To date, the ALEGRA code has not developed a technique to return the tensor to SPD. The results of a non-SPD tensor can be devastating and thus a search for a correction to this feature continues. Lastly, care must be taken in advecting other variables that have special properties. For example, a parameter which can only be one or zero must remain one or zero after the advection.

3.2.5 Vertex Centered Advection

Vertex centered advection is very similar in concept to element centered advection and is required for advecting nodal quantities such as momentum. In vertex centered advection, two approaches can be used - staggered mesh or element based. Many authors have used a staggered grid or median like mesh. A staggered grid is developed by connecting the element and edge centers as shown in Figure 28 for a quadrilateral mesh. With this mesh, the nodal points are at the center of a volume formed by the staggered grid. This grid forms eight faceted polygons in quadrilaterals and thirty-two faceted polygons in hexahedrons. Although this approach is straight forward, it requires the calculation and advection of quantities through each facet. For ALEGRA it was believed that this approach is too costly in memory and computational requirements.

The element based approach uses the fluxes already calculated for the elements and uses the advection stencils already developed for the element based quantities. There are three element based vertex centered advection methods in ALEGRA: Simplified ALE[15] (SALE), Half Interval Shift[16] (HIS) and Modified HIS. All production calculations currently use the HIS method, but all three of these methods will be described below.

The SALE method uses an averaging approach. Vertex centered values are averaged at the element centers. These averaged quantities are advected and the newly advected element quantities are averaged back to the nodes. Although not used in ALEGRA, the SHALE[17] method improves on SALE by also advecting the derivative of the vertex quantity. Computationally, SALE is very inexpensive relative to staggered methods and HIS but is not monotonic and very diffusive. The SALE method is in ALEGRA for historical reasons and is only used for verification problems in order to demonstrate the effects of less accurate methods.

ALEGRA's production vertex centered advection scheme is Benson's Half Interval Shift algorithm[16]. Although Benson developed this scheme for logically rectangular grids and

proved the method to be second order and monotonic, the method has been extended in ALEGRA for unstructured grids. The HIS method is very attractive because it uses the infrastructure developed for element centered quantities. The HIS method begins by considering vertex quantities for an element to be considered at the center of the element. This is shown in Figure 29. Each vertex quantity is then advected with the element centered advection methods and the change in the vertex quantity is summed (assembled) at the vertex. The summed quantity, Γ is given as

$$\Gamma = \sum_{i=1}^{ne} \tilde{f}_i(\Delta M_{ni} - \Delta M_i) = \sum_{i=1}^{ne} \Gamma_i \tag{45}$$

where *ne* is the number of elements attached to a vertex. In applying this method, each element can be processed in a finite-element approach, wherein all of an element's vertices are processed and the result is assembled at the vertex. After all elements are processed, the new vertex value is given as

$$f_{new} = \frac{f_{old} M_{old} + \Gamma}{M_{new}} \tag{46}$$

One of the difficulties in applying the HIS algorithm is determining the "correct" ahead and behind values for Van Leer or SuperB advection schemes. The Modified HIS algorithm uses an element averaged value. This method, although easy to implement, has been found to be much less accurate than the original HIS algorithm. In the original HIS algorithm, a logically rectangular mesh automatically determines the upstream and downstream vertices and thus the values to use in the second order advection schemes. In an unstructured mesh, the vertices and their values are less certain, since a vertex can have any number of vertex neighbors.

In ALEGRA, the HIS algorithm has been applied by considering the upstream and downstream elements and finding the vertices that lie along an edge with the vertex. Due to the various element orientations that are produced by grid generation tools, these vertex "neighbors" must be determined for each vertex along each edge, as shown in Figure 29. Due to memory concerns, ALEGRA finds these neighbors dynamically using the element neighbor information and face-node lists. This approach requires an insignificant amount of computation as compared to the second order advection calculation and greatly improves the solution as compared to the Modified HIS method.

ALEGRA allows for vertex center quantities to be advected with mass or volume. In addition, in two-dimensional axisymmetric calculations, quantities can be advected with the element's area. The choice of the advection weight depends on the quantity being advected and is generally chosen to conserve a global variable. For example, since momentum is the product of mass and velocity, the advection of momentum is based on mass. This also ensures that the DeBar's consistency condition[18] is satisfied. In simple terms, the DeBar condition requires that uniform velocity remain unchanged by the advection algorithm.

3.2.6 Viper Shaped Charge

An example of the improved results that are delivered by the HIS advection scheme, as opposed to the Modified HIS scheme, can be seen in the simulation of a shaped charge. This device, called a Viper shaped charge generator, consists of a conical copper liner surrounded by high explosive (HE). The initial configuration of the model is shown in Figure 30. The HE is point detonated at the base of the assembly and the expanding detonation wave causes the copper liner to collapse upon itself, resulting in a jet of molten copper being ejected along the axis of the device. This problem was modeled with ALEGRA's HIS advection and the Modified HIS advection to explore the difference in the results. As can be seen in Figure 31, even at a relatively early time of 35 μs there is a distinct difference in the length of the jet formed. By a time of 87.5 μs, as shown in Figure 32, the difference in length is about 5 cm. The base of the jet using the two models is very similar, so the longer extension of the HIS-modeled jet accounts for the better agreement with data.

4. Conclusion

The MMALE method developed in ALEGRA has proven to be a very robust approach to solving problems with large strain rates. This method has been extended beyond shock physics and is currently used on simulations requiring magnetohydrodynamics, electrostatics and radiation transport. By using the concept of pseudo one-dimensional passes, the traditional second order structured methods of interface reconstruction and element and vertex advection have been extended to unstructured grids. These algorithms have been shown to demonstrate second-order behavior on unstructured grids. However, work still remains in effectively remeshing complex deformed boundaries.

References

1. Summers, R. M., Peery, J. S., Wong, K. W., Hertel, E. S., Trucano, T. G., and Chhabildas, L. C.,Recent Progress in ALEGRA Development and Application to Ballistic Impacts, *International Journal of Impact Engineering*, to be published, 1997.

2. Benson, D. J. Computational Methods in Lagrangian and Eulerian Hydrocodes, *Computers Methods in Applied Mechanics and Engineering*, **99**, 1989.

3. Benson, D. J., An Efficient Accurate, Simple ALE Method for Nonlinear Finite Element Programs, *Computers Methods in Applied Mechanics and Engineering*, **72**, 1989.

4. Sharp, R. W., and Barton, R. T., "HEMP Advection Model," UCID-17809 Rev. 1, Lawrence Livermore National Laboratory, Livermore, CA, 1981.

5. Abramowitz, M. and Stegun, I. A., Handbook of Mathematical Functions, Dover Publications, New York, ISBN 0-486-61272-4, 1970.

6. Budge, K. G., ALE Shock Calculations Using a Stabilized Serendipity Rezoning Scheme, *Shock Compression of Condensed Matter*, 1991.

7. Winslow, A. M., "Equipotential Zoning of Two Dimensional Meshes," UCRL-7312, Lawrence Livermore National Laboratory, Livermore, CA, 1963.

8. Tipton, R. E., Private Communication, Lawrence Livermore National Laboratory, Livermore, CA, 1991

9. Pert, G. J., Physical Constraints in Numerical Calculation of Diffusion, *J. Comp. Phys.*, **42**, 1981.

10. VanLeer, B. Towards the Ultimate Conservative Difference Scheme V. A Second-Order Sequel to Godunov's Methods, *J. Comp. Phys.*, **32**, 1979.

11. Roe, P. L., Some Contributions to the Modeling of Discontinuous Flows, *Lectures in Applied Mathematics*, **22**, 1985.

12. Noh, W. F. and Woodward, P., SLIC (Simple Line Interface Calculations, *Lecture Notes in Physics*, **59**, Springer Verlag, 1976.

13. R.L. Bell, E.S. Hertel, "An Improved Material Interface Reconstruction Algorithm for Eulerian Codes," SAND 92-1716, Sandia National Laboratories, Albuquerque, NM, 1992.

14. Mcglaun, J. M., Thompson, S. L., Kmetyk, C. N., and Elrick, M. G., “A Brief Description of the Three-Dimensional Shock Wave Physics Code CTH,” SAND89-0607, Sandia National Laboratories, Albuquerque, NM, 1990.

15. Amsden, A. A., Ruppel, H. M. and Hirt, C. W., “SALE: A simplified ALE Computer Program for FLuid Flow at All Speeds,” LA-8095, Los Alamos National Laboratory, Los Alamos, NM, 1980.

16. Benson, D. J., Momentum Advection on a Staggered Mesh, *J. Comp. Phys.*, **100**, 1992.

17. Demuth, R. B., et. al., “SHALE: A Computer Program for Solid Dynamics,” LA-10235, Los Alamos National Laboratory, Los Alamos, NM, 1985.

18. Debar, R. R., “Fundamentals of the KRAKEN Code,” Lawrence Livermore Report UCIR-760, Lawrence Livermore National Laboratory, Livermore, CA, 1974.

19. Attaway, S. W., “Update of PRONTO 2D and PRONTO 3D Transient Solid Dynamics Program,” SAND90-0102, Sandia National Laboratories, Albuquerque, NM, 1990.

Acknowledgment

The ALEGRA code is being developed in the Computational Physics Research and Development Department (9231) at Sandia National Laboratories. At its inception, ALEGRA used the PRONTO[19] code as its foundation for Lagrangian transient dynamics. ALEGRA has evolved to solve a new class of problems that involve strong shocks and intense energy deposition. For this class of problems, Multi-Material Arbitrary Lagrangian-Eulerian (MMALE) methods were added. The authors of the ALEGRA code are indebted to the authors of PRONTO for providing a superior framework to begin development. The rapid progress made to date on ALEGRA could not have occurred without this well thought out framework.

Nomenclature

ALE	Arbitrary Lagrangian Eulerian
ALEGRA	Arbitrary Lagrangian Eulerian General Research Applications code
ICF	Inertial Confinement Fusion
MMALE	Multi-Material ALE
SMALE	Single-Material ALE
SMYRA	Sandia's Modified Youngs Reconstruction Algorithm
HIS	Half Interval Shift
SLIC	Simple Line Interface Construction

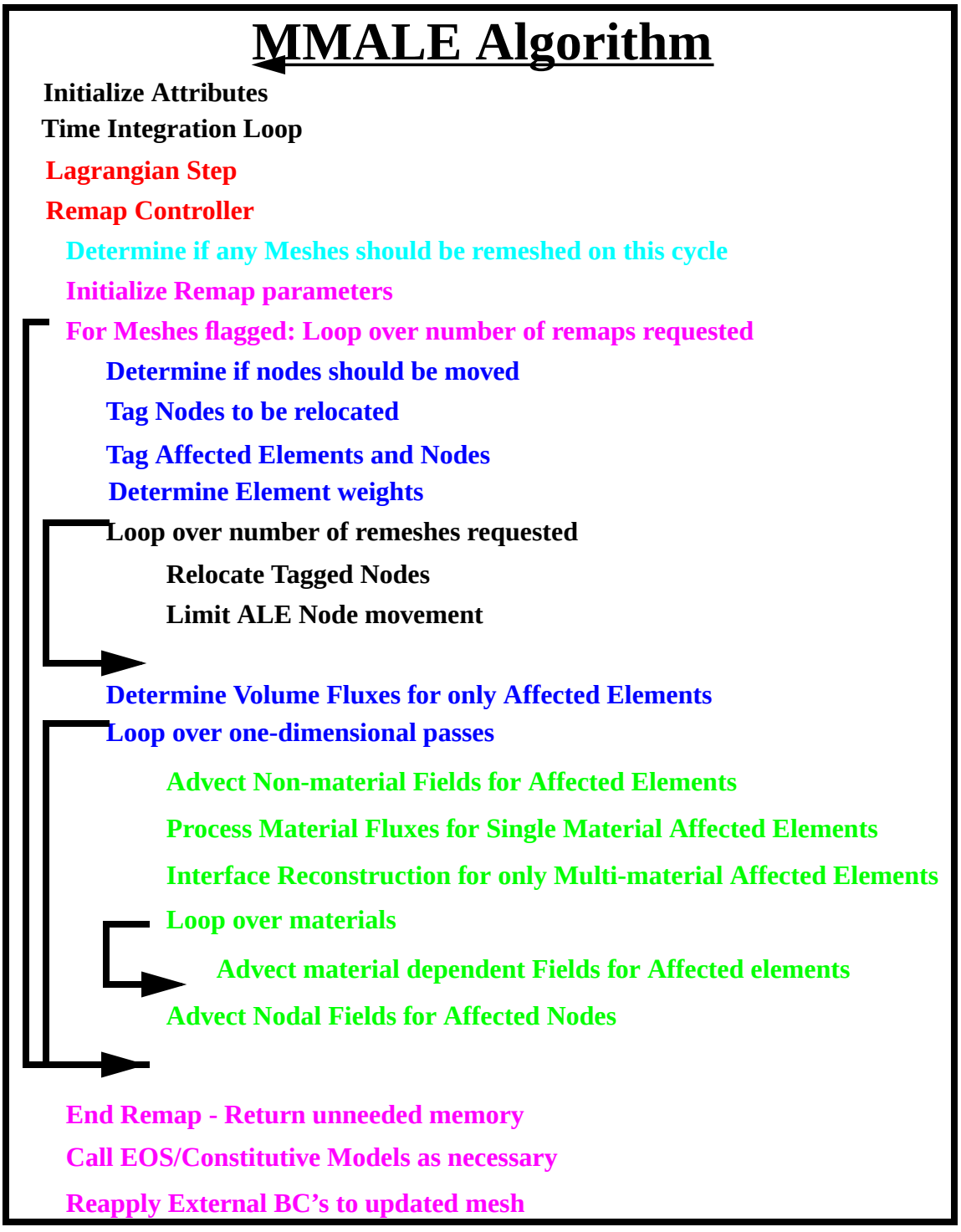


Figure 1 Flowchart for MMALE package in ALEGRA

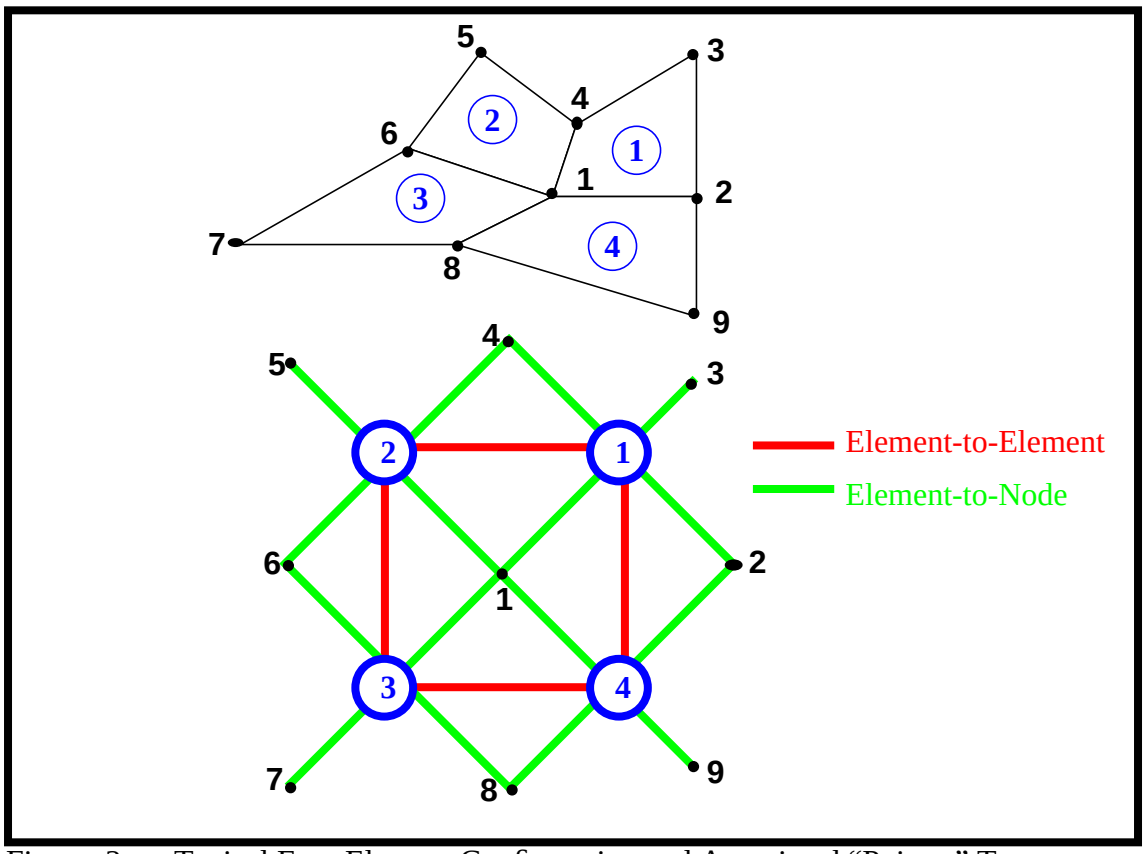


Figure 2 Typical Four Element Configuration and Associated “Pointer” Tree.

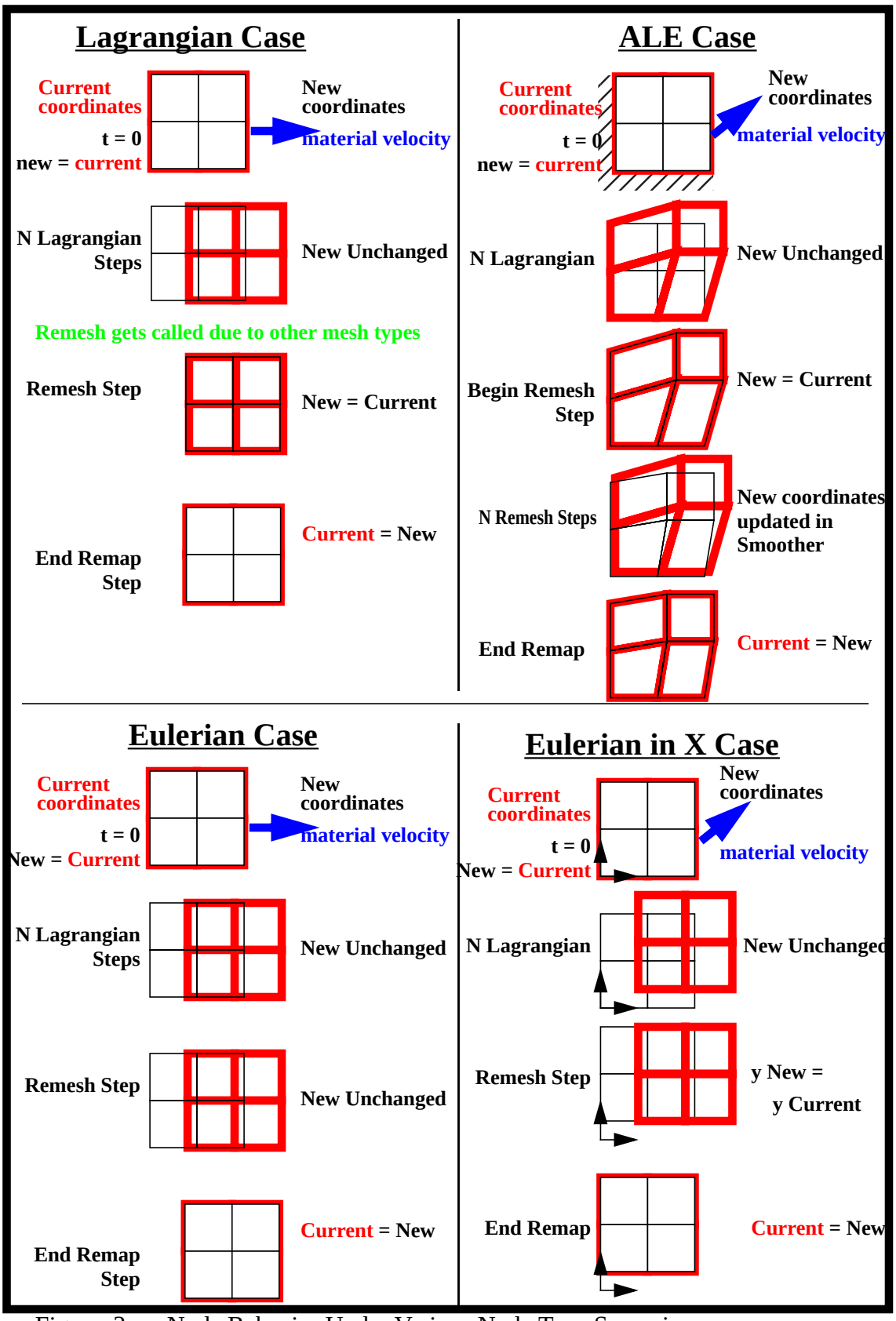


Figure 3 Node Behavior Under Various Node Type Scenarios

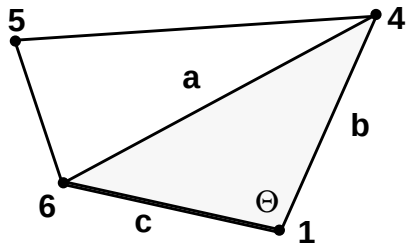


Figure 4 Typical Quadrilateral Element and Associated Angle at the Reference Node

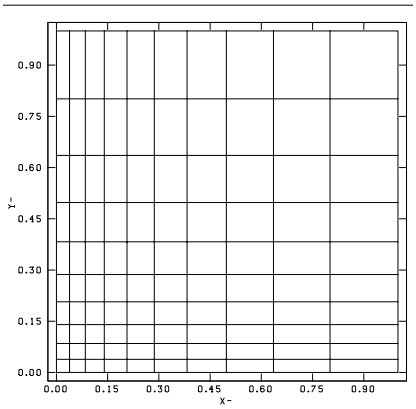


Figure 5 Initial 10x10 Graded Mesh

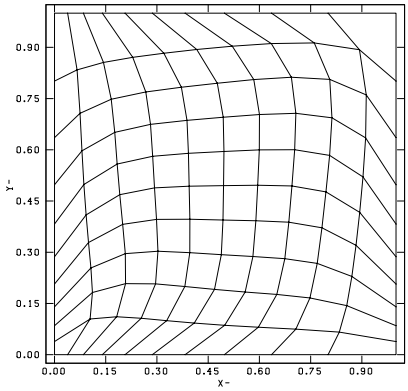


Figure 6 Tipton Smoother at 10 cycles with 10 iterations per cycle

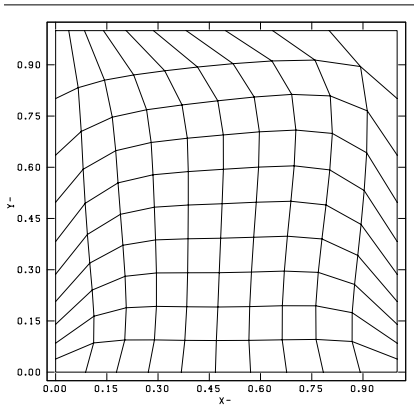


Figure 7 Tipton Smoothing with an ALE node set at Y=0 at 10 cycles with 10 iterations per cycle.

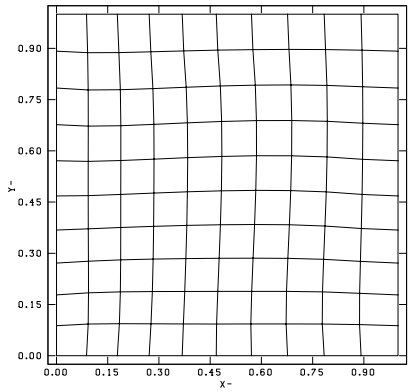


Figure 8 Tipton Smoothing with all ALE Boundaries.

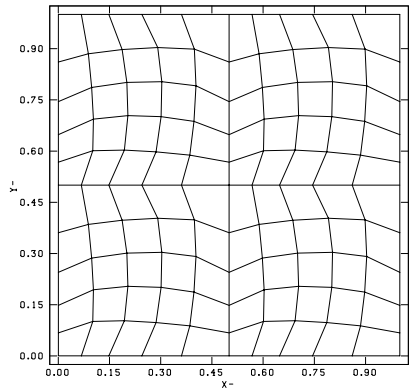


Figure 9 Behavior of Interface Nodes in a Four SMALE Mesh at 10 cycles with 10 iterations per cycle.

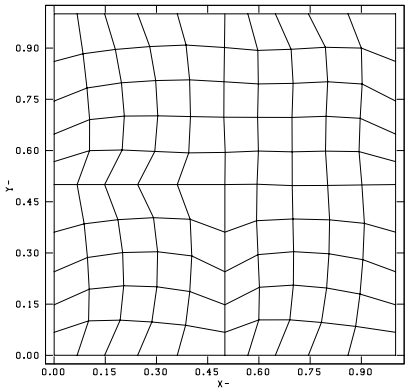


Figure 10 Behavior of Interface Nodes between a SMALE Mesh and three MMALE meshes.

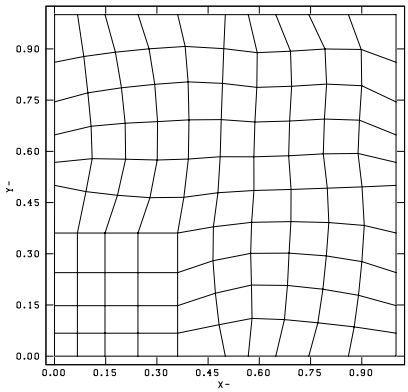


Figure 11 Behavior of Interface Nodes between a EULERIAN Mesh and three MMALE meshes.

Acknowledgment

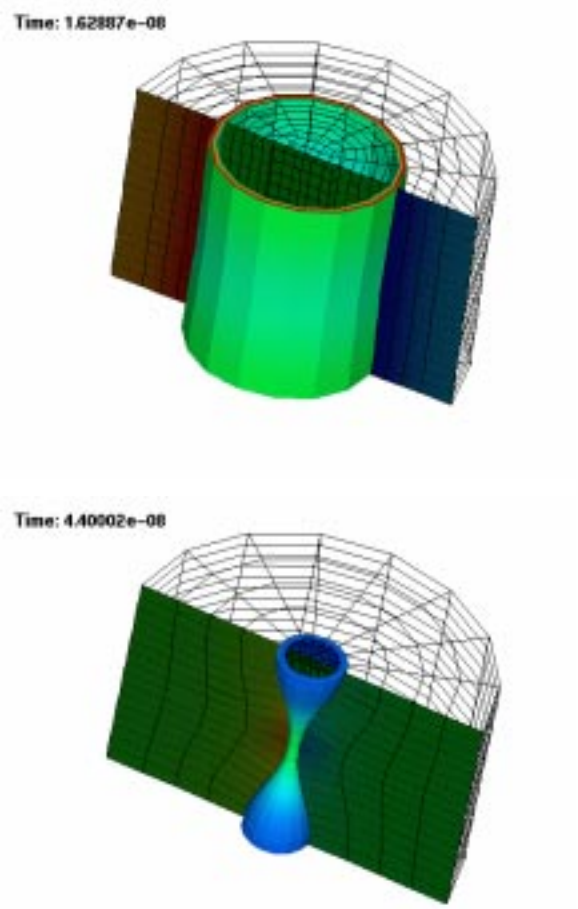


Figure 12 Three Dimensional Tipton Smoothing with inverse radius weighting

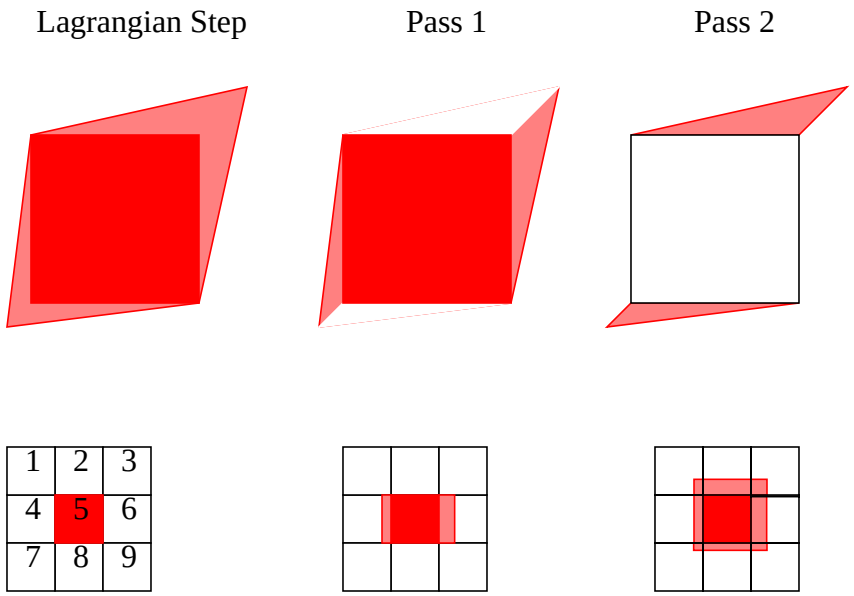


Figure 13 The effect of one-dimensional passes in the advection stencil

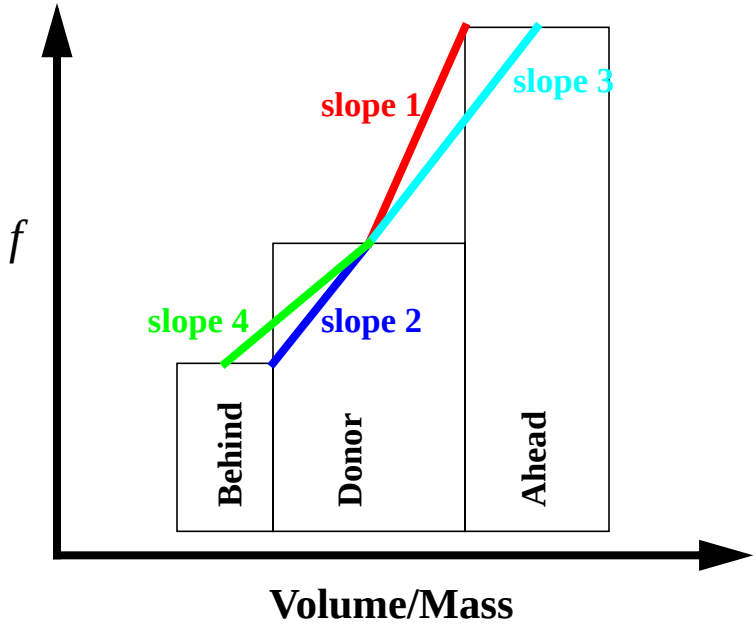


Figure 14 Advection Slopes for Second Order VanLeer Scheme

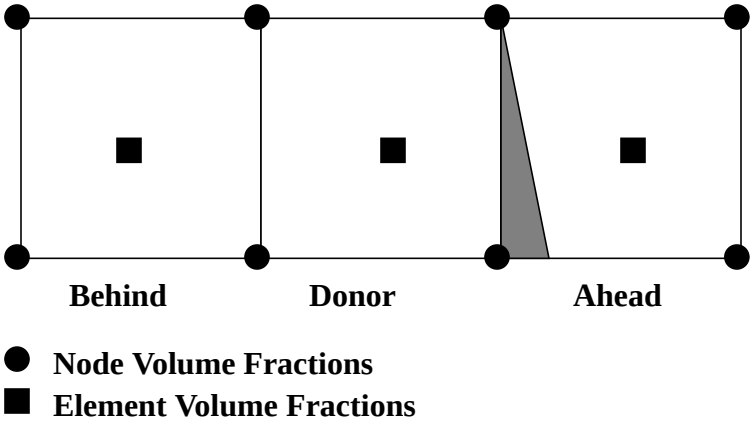


Figure 15 SMYRA Advection Stencil - Required Volume Fractions for Donor Material Flux Determination

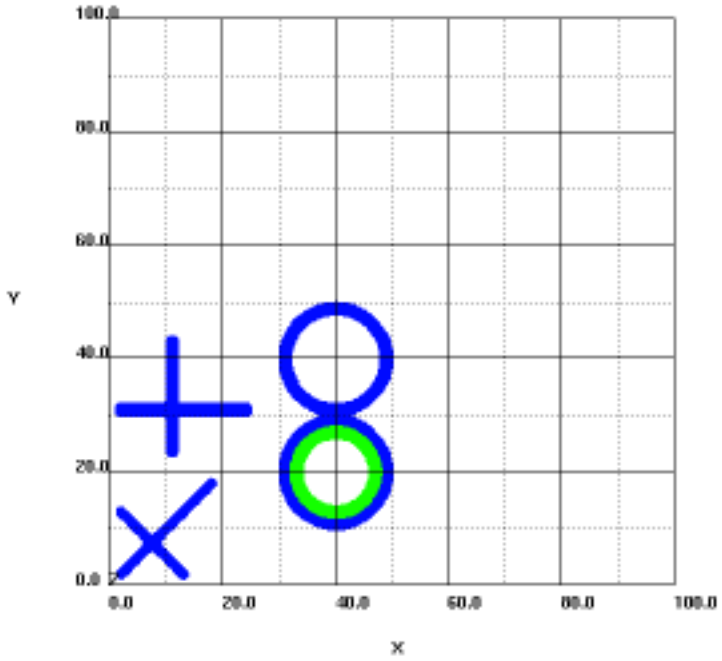


Figure 16 Two-Dimensional Ball and Jacks Initial Conditions

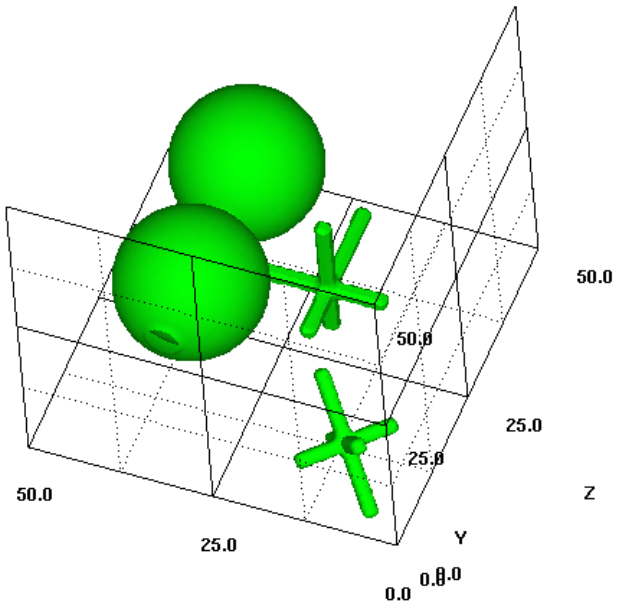


Figure 17 Three-Dimensional Ball and Jacks Initial Conditions

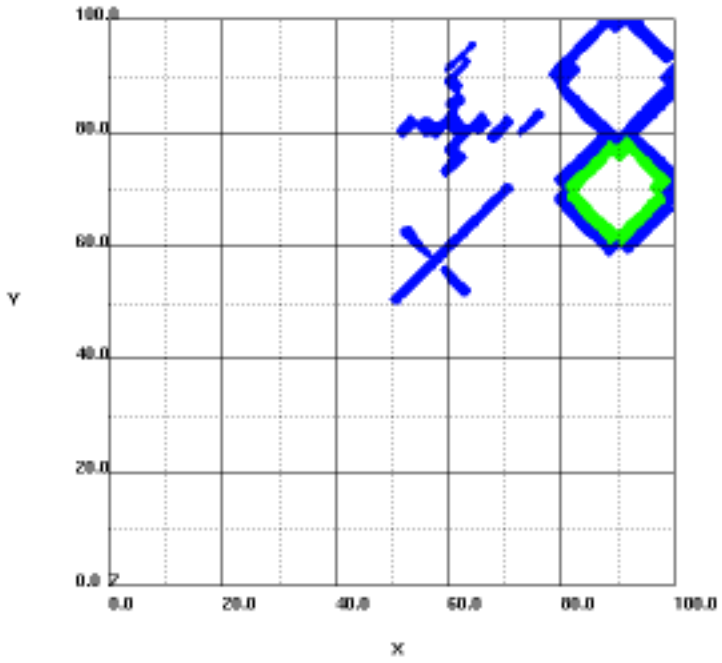


Figure 18 Final State of SLIC solution of Two-Dimensional Ball and Jacks on a structured 40,000 cell grid

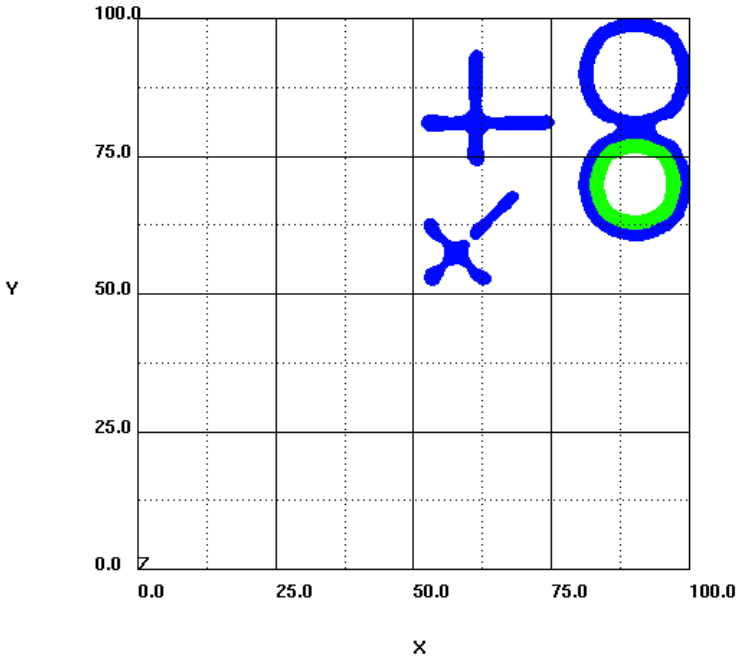


Figure 19 Final State of SMYRA solution of Two-Dimensional Ball and Jacks on a structured 40,000 cell grid

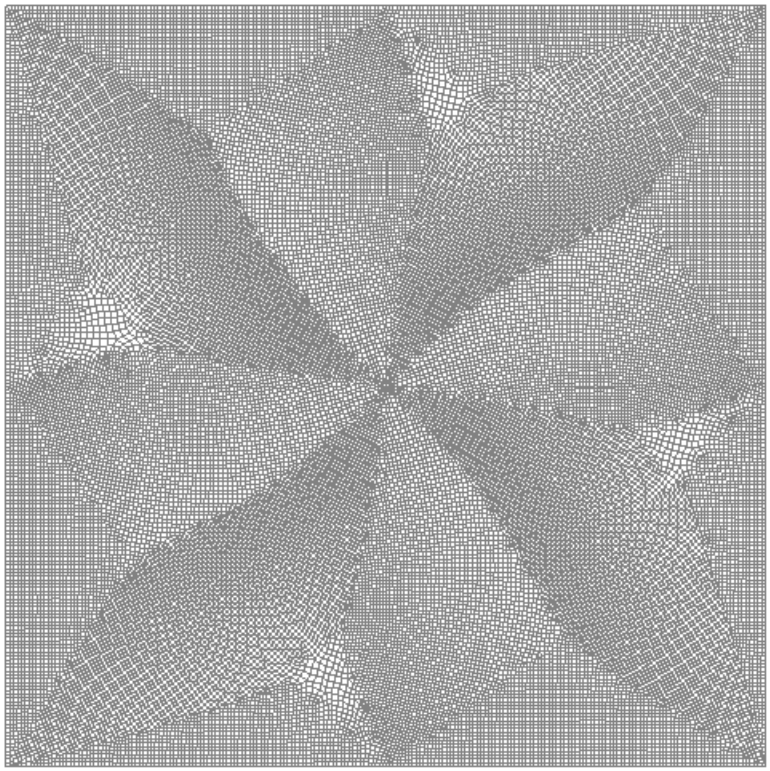


Figure 20 An unstructured ~40,000 cell grid used with the two-dimensional Balls and Jacks examples.

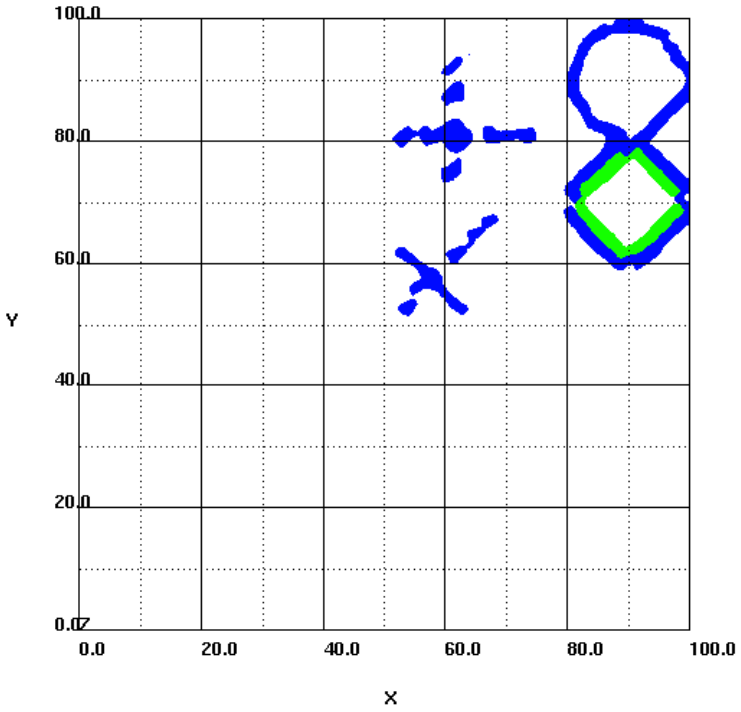


Figure 21 Final State of SLIC solution of Two-Dimensional Ball and Jacks on a unstructured ~40,000 cell grid with mesh shown.

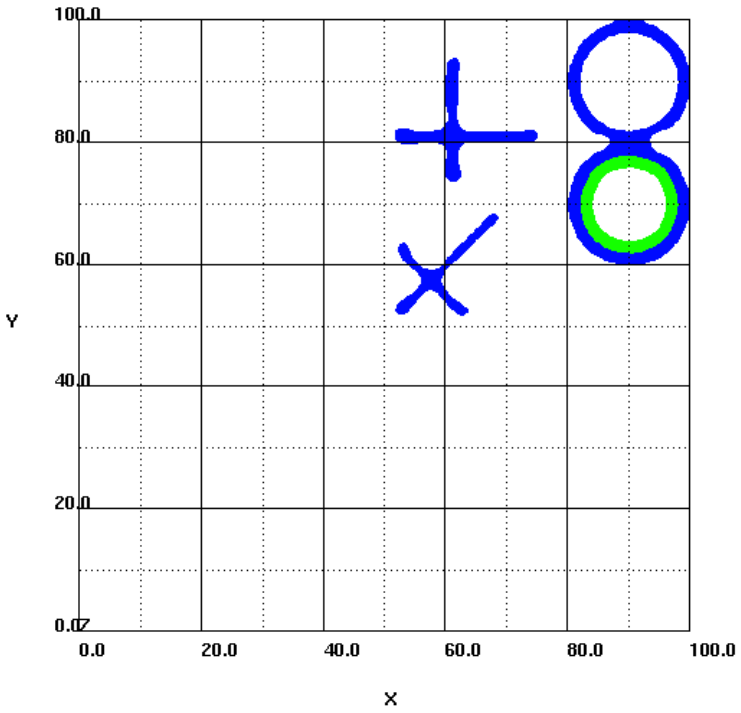


Figure 22 Final State of SMYRA solution of Two-Dimensional Ball and Jacks on a unstructured ~40,000 cell grid.

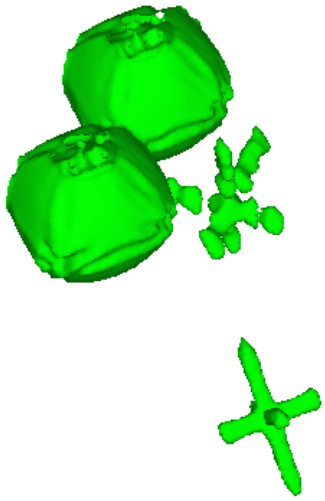


Figure 23 Final State of SLIC solution of Three-Dimensional Ball and Jacks on a structured 1 million cell grid.

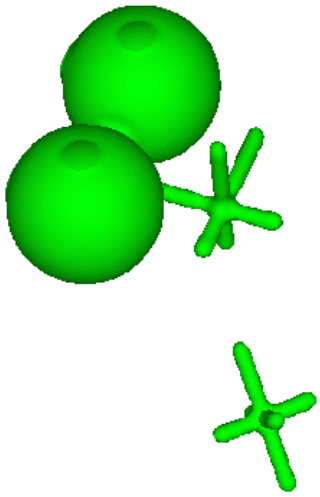


Figure 24 Final State of SMYRA solution of Three-Dimensional Ball and Jacks on a structured 1 million cell grid.

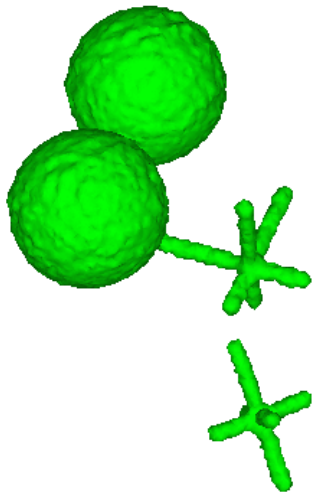


Figure 25 Final State of SLIC solution of Three-Dimensional Ball and Jacks on a unstructured 1.1 million cell grid.

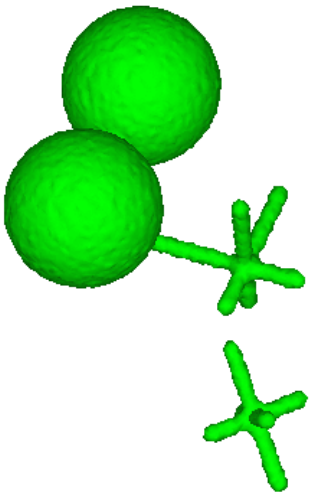


Figure 26 Final State of SMYRA solution of Three-Dimensional Ball and Jacks on a unstructured 1.1 million cell grid.

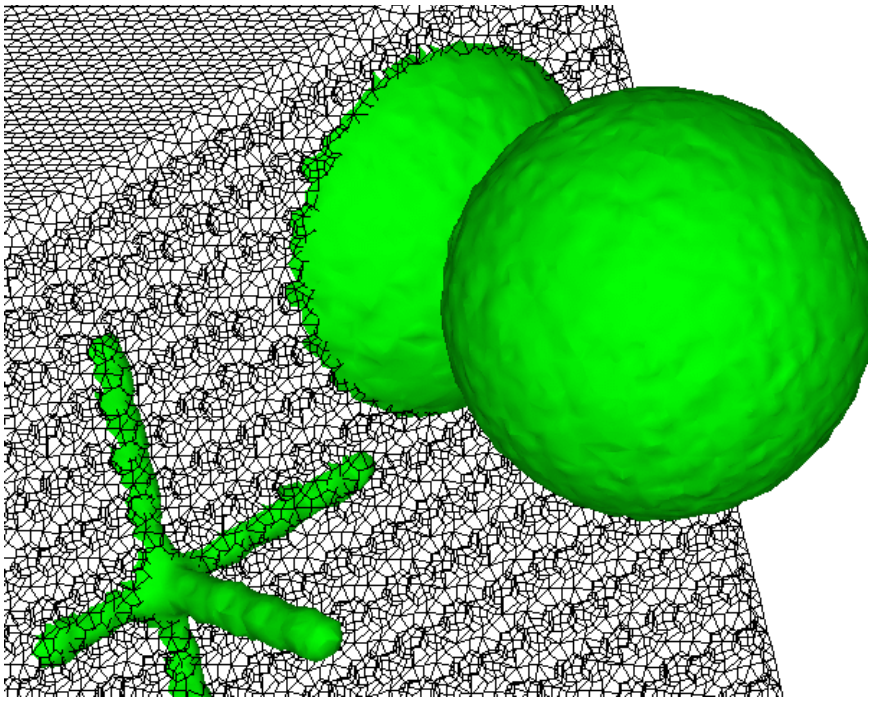


Figure 27 Final State of SMYRA solution of Three-Dimensional Ball and Jacks on an unstructured 1.1 million cell grid with the objects shown extruding from the three dimensional mesh.

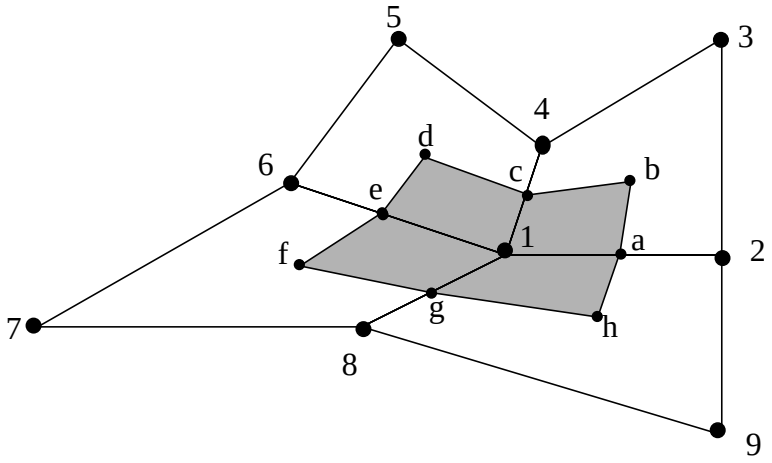
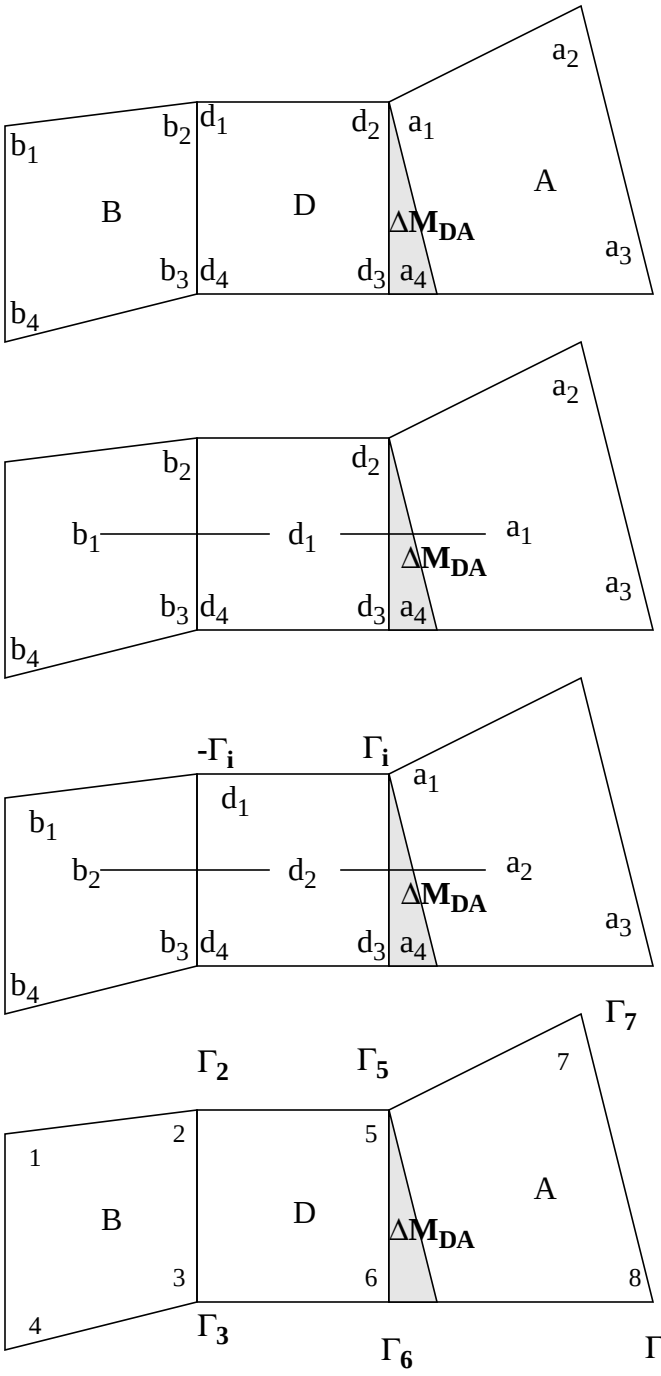


Figure 28 Typical staggered grid for vertex centered advection



Given an advection mass, $\Delta\mathbf{M_{DA}}$ from D to A, each vertex value is considered in each element. In this picture $d_2 = a_1$ etc.

Starting with Element D and its first vertex d_1, neighboring vertices, b_1 and a_1 must be determined for the advection stencil. For unstructured grids, this requires element neighbor and orientation information.

Using an element centered advection stencil, Γ_i is determined and summed at locations d_1 and a_1. Next, position 2 is processed followed by the remaining positions.

After processing Element D, new partial momenta have been summed at the six globally numbered vertices involved with the advection mass $\Delta\mathbf{M_{DA}}$. This continues until all elements have been processed.

Figure 29 Steps in the Half Interval Shift algorithm for unstructured grids.

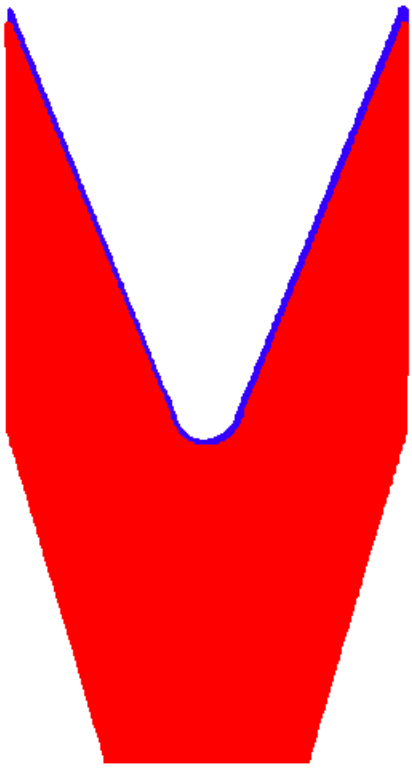


Figure 30 Shape Charge Initial Conditions

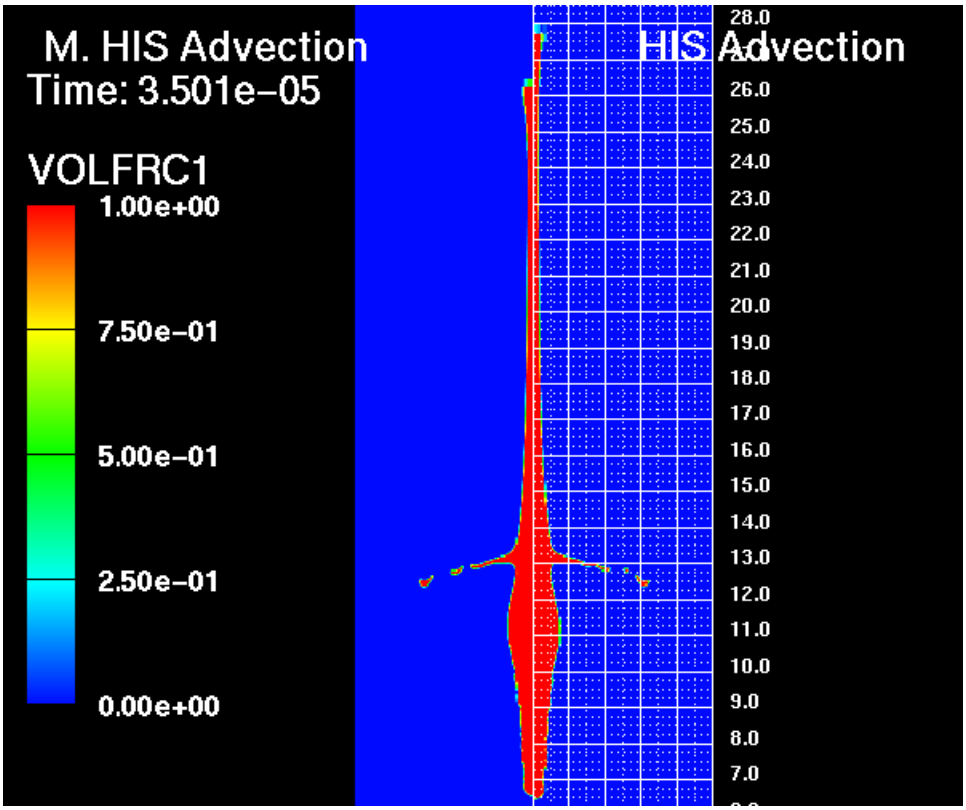


Figure 31 Comparison of HIS and MHIS Advection schemes for VIPAR Shape Charge at 35 μs.

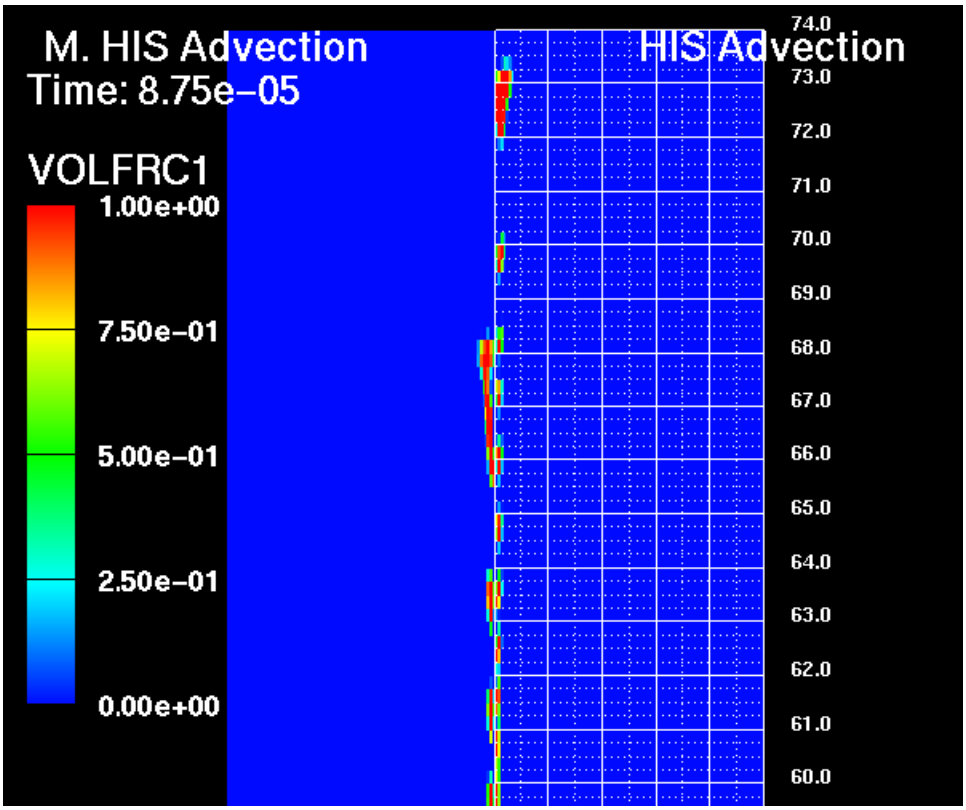


Figure 32 Comparison of HIS and MHIS Advection schemes for VIPAR Shape Charge at 87.5 μs.